Database systems

A Database System is:

- essentially a computerized record keeping system.
- purpose: to maintain information and make it available as required.

Facilities provided to the user of a database

- Adding new (empty) files to the database
- Inserting data into existing files
- Retrieving data from existing files
- Updating data in existing files
- Deleting data from existing files
- Removing files from the database

Some Terms

Database: a logically coherent collection of data with some inherent meaning; persistent.

Database Management System: collection of programs that enables users to create and maintain a database. A *general purpose* software system that facilitates the process of defining, constructing and manipulating database for various applications.

Data Definition Language (DDL): used to define types, access structures, constraints.

Data Manipulation Language (DML): used to query or modify the data.

A Brief History of Databases

1950's - ad-hoc techniques for storing and accessing data.

1961 - First generalized database system, based on the network model.

1960's - File oriented view of databases, hierarchical and network models.

1970 - relational model formalized by Codd

1976 - ER (entity-relation) model formalized by Chen

1970's - development of many prototype relational systems and query languages (e.g. SQL).

1980's - commercial relational systems, PC-based database systems, SQL becomes de facto standard, research on object-oriented database systems and deductive database systems.

1990's - (Predictions) Emphasis on distributed databases, object-oriented databases, knowledge-bases, special-purpose databases (e.g. spatial, temporal, multi-media, text-oriented),

A Basic Model

- Computerized files are also called **tables**
- Rows of a table are known as **logical records** or just **records**
- Columns of a table are known as **fields**
- **SELECT, INSERT, UPDATE,** and **DELETE** are operations of SQL (Structured Query Language - to be discussed later).

Database System Components

A database System consists of data, hardware, software, and people.

Data - *integrated* from the requirements of many applications (eliminating redundancy) and *shared* by many users who use it in different ways.

Hardware - The hardware components of the system may include:

- the processor or processors
- main memory
- persistent storage (e.g., disks, CD-ROM, etc.)

Database System Components

Software

- a **database manager**, or a **database management system (DBMS)**, handles all requests made by the user to access the database, and shields the underlying hardware details from the users of the system.
- Other software components of a database, e.g., Utilities, Application development tools, Design aids, Report writers, etc.m

People:

- Data Administrator (DA)
- Database Administrator (DBA)
- Application programmers
- End users (vary by type of interface Menu- or form-driven vs. Command-driven)

Types of Data

A database consists of some collection of persistent data that is used by the application systems of some given enterprise.

Persistent Data - Data in a database is usually referred to as *persistent data*. It is so called because it is different from the input data, output data, control statements, work queues, software control blocks, intermediate results etc.

Input Data - Data entering the system for the very first time (from a keyboard, for example). It might cause a change to the persistent data.

Output Data - Messages and results coming out of the system (those printed on the screen). It is derived from the persistent data.

Entities and Relationships

A manufacturing company will typically, have

- Projects on hand
- Parts used in those projects
- Suppliers who supply those parts
- Warehouses in which parts are stored
- Employees who work on the projects, etc.

Nouns, e.g., projects, parts suppliers, usually correspond to *entities*.

Verbs, e.g., supply, "work on", etc, usually correspond to *relationships*.

Entities and Relationships

Entity - any distinguishable object that is to be represented in the database. Distinguish types and instances.

Relationships - relationships linking the entities, represented by connecting lines or arcs.

- taken as bi-directional (e.g. Who supplies widgets?, What does Acme supply?)
- may be binary, ternary or of any arity ¿ 1 (what are 0 and 1 arity relations?)
- there may be several relationships between the same two entities (e.g. employees are assigned to projects, employees manage projects)

Properties - Entities and hence Relationships have properties. Usually simple and can be represented by simple data types.

Why are Databases used

Advantages over manual record keeping

- **Compactness:** No need of large paper files.
- **Speed:** A machine can retrieve and change data faster than a human.
- **Less Drudgery:** Much of the sheer tedium of maintaining files by hand is eliminated. Mechanical tasks are always better done by machines.
- **Currency:** Accurate, up-to-date information is available on demand at any time.

The database system provides the enterprise with *centralized* control of its data.

Data Administration and Database Administration

Data Administrator (DA)

- Person with central responsibility for the data
- Decides what data should be stored in the database
- Establishes policies to maintain that data

Database Administrator (DBA)

- Person who implements the DA's decisions
- Creates the actual database
- Ensures that the system's performance requirements are met
- Ensures that the system provides the variety of services it has been designed for.

Benefits of the Database Approach

Redundancy can be reduced

- *Usually* there is only one copy of all *files* in a database system
- All redundancy need not be necessarily eliminated
- In case of multiple copies updates should be *propagated* to all copies.

Inconsistency can be avoided

- When two copies of the same data object differ, they are inconsistent
- If redundancy is eliminated then inconsistency does not arise
- If redundancy is controlled then the user never gets an inconsistent view of the database

Benefits of the Database Approach

Centralized data maintenance

- Updating
- Backup
- Recovery

Conflicting requirements can be balanced

- DBA can structure the database such that it provides an overall service which is best for the enterprise

Benefits of the Database Approach

Data can be shared

- Existing as well as new applications can share the same data
- No additional data need be created for new applications.

Standards can be enforced

- DBA can enforce standards in the representation of the data
- Desirable for migration of data between systems
- Data naming and documentation standards aid in data sharing and understandability.

Data Independence

Data independence is defined as the immunity of applications to change in storage structure and access technique.

A DBMS system should provide the user with a **conceptual representation** of the data that does not include (many) of the details of how the data is stored or is to be accessed.

- A data dependent system is one in which the knowledge of data organization and access technique is built into the application logic and code.

Data Independence is important to:

- insulate programs from data for maintainability.
- support multiple data views and requirements for flexibility

Benefits of the Database Approach

Security restrictions can be applied

- DBA can ensure access to database through proper channels
- DBA can define security checks to be carried out whenever access is attempted to sensitive data

Integrity can be maintained

- Integrity implies that the values of the data objects are accurate
- Even without redundancy the database might contain incorrect data
- Centralized control can take of this problem
- Becomes very important in a multi-user environment.

What might change in a representation?

Typical aspects of the database storage structure subject to change:

- **Representation of numeric data**
  DBA must choose an appropriate base (binary or decimal), scale (fixed or floating point), mode (real or complex), and precision (number of digits). Any of these may change to improve performance or to conform to new standard etc.
- **Representation of character data**
  A character string might be stored in any of several distinct character codes (ASCII, EBCDIC).
- **Units of numeric data**
  The units in a numeric field might change
- **Data coding**
  In some situations it might be desirable to store data by coded values.

- **Data materialization**
  Direct: An occurrence of a logical field has a corresponding stored field.
  Indirect: The value of a logical field is materialized by performing some computation on a number of stored fields.
- **Structure of stored records**
  Two existing records might be combined into one or a single record might be split into two.
- **Structure of stored files**
  A given stored file can be physically implemented in storage in a wide variety of ways. For example, it might be entirely contained within a single storage volume, or might be spread across several volumes on different device types.

Enabling the database to grow without logically impairing existing applications is probably the most important reason for requiring data independence.

Relational Systems And Others

Relational Approach

- The data is perceived by the user as tables
- The operator's at the users disposal are operators that generate new tables from old.

Some recent approaches

- Deductive DBMS's
- Semantic DBMS's
- Universal relation DBMS's
- Object-oriented DBMS's
- Extensible DBMS's
- Expert DBMS's

Relational Model

The Relational Model: one structure, the relation (table), which is a *set* of tuples or mappings from attribute names to values in the underlying domain of the attribute.

R(A_1, A_2,...,A_n) will denote the relation named R with *scheme* (set of attributes) A_1, A_2,...,A_n.
Example: PERSON(LName, FName, Address, City, Zip)

We will use the following notation for an *instance* of R:

R :

A_1	A_2	...	A_n
...	...	...	...

The columns denote *attributes*, the rows *tuples*.
The *size* of a relation is the number of tuples

Relational Model - Example

Example: Let
D_1= {0, 1} be the underlying domain of NUM;
D_2= {a, b, c} be the underlying domain of VAL.

Then the following is a valid relation over the scheme NUM, VAL:

R :

NUM	VAL
0	a
1	b
0	b

$\equiv$

NUM	VAL
1	b
0	a
0	b

$\equiv$

VAL	NUM
b	1
a	0
b	0

size(R)=3, arity(R)=2.

Relational Model - DMLs

Relational DML's: based on the *relational algebra* or *relational calculus* (which are computationally equivalent).

Operands: constant relations or relational expressions.

Operators:

- **unary** - select (σ), project (Π);
- **binary** - union ($\cup$), set difference (-), cartesian product ($\times$), ...

Relational Model - Unary Operators

Unary Operators:

R :

A	B
a_1	b_1
a_1	b_2

1. Select: σ_F(R), where F is a formula involving:
   – operands that are constants or attribute names
   – arithmetic comparison operators ($<$, $>$, $=$, $\geq$, $\leq$, $\neq$)
   – logical operators ($\vee$, $\wedge$, $\neg$)

   Horizontal subset
   $\sigma_{B='b_1'}$(R) :

A	B
a_1	b_1

2. Project: $\Pi_{A_1,A_2,\ldots,A_k}$(R)

   Vertical subset
   Π_A(R) :

A
a_1

Relational Model - Binary Operators

R :

A	B
a_1	b_1
a_1	b_2

S :

A	B
a_1	b_1
a_2	b_1
a_2	b_2

1. Union

   R$\cup$S :

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2

2. Set difference

   R-S :

A	B
a_1	b_2

3. Cartesian product

$R \times S$:

R.A	R.B	S.A	S.B
a_1	b_1	a_1	b_1
a_1	b_1	a_2	b_1
a_1	b_1	a_2	b_2
a_1	b_2	a_1	b_1
a_1	b_2	a_2	b_1
a_1	b_2	a_2	b_2

4. Intersection: $R \cap S \equiv$ R-(R-S)

$R \cap S$:

A	B
a_1	b_1

5. θ-join: R $\theta_{i<rel-op>j}$ S

$R\theta_{R.A \neq S.A}S$:

R.A	R.B	S.A	S.B
a_1	b_1	a_2	b_1
a_1	b_1	a_2	b_2
a_1	b_2	a_2	b_1
a_1	b_2	a_2	b_2

6. Natural Join: R $\bowtie$ S

Common attributes must have equal values; attributes appear only once.

S :

A	B
a_1	b_1
a_2	b_1
a_2	b_2

T :

B	C
b_1	c_1
b_1	c_2

$S \bowtie T$:

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_2

Relational Model - EXAMPLES

STUDENT(FName, LName, SNum, Address, Major)
COURSE(CNum, CName, Desc)
FACULTY(FName, LName, SNum, Office, OPhone)
TEACHES(SNum, CNum, Semester, Year)
ENROLLMENT(SNum, CNum, Semester, Year)

1. "Print all information about students."

   STUDENT
2. "Print the LName of all students."

   Π_{LName} STUDENT
3. "Print all student information about CS majors."

   $\sigma_{Major='CS'}$STUDENT
4. "Print the LName of all CS majors."

   $\Pi_{LName}(\sigma_{Major='CS'}$STUDENT)

5. "Print the SNum of all students who are not also professors."
   (Π_{SNum}STUDENT)-(Π_{SNum}FACULTY)
6. "Print the LName of all students in CS100 during Fall 1991."

   Π_{LName}(STUDENT $\bowtie$ ($\sigma_{CNum='CS100' \wedge Semester='Fall' \wedge Year=1991}$ENROLLMENT))
7. "Print the SNum of all Students who took some course with Davidson during 1991."

   Let T= ($\sigma_{LName='Davidson'}$ FACULTY) $\bowtie$ ($\sigma_{Year=1991}$ TEACHES) in

   Π_{SNum}(ENROLL $\bowtie$ T)